AI-Assisted Zero-Trust Optimization for Energy-Efficient Microservices in Financial Systems

Muzeeb Mohammad
Georgia Institute of Technology
Atlanta, Georgia 30332, USA

ABSTRACT
Financial institutions increasingly rely on cloud-native microservices to deliver low-latency trading, payments, and risk-analysis platforms. While this architecture improves agility and scalability, it simultaneously amplifies authentication overhead, security risk exposure, and overall energy consumption. Earlier work by the author introduced a performance-optimized Zero-Trust API security model using token-less authentication and optimized mutual TLS (mTLS), while a companion study demonstrated that asynchronous communication and ARM-based serverless/container runtimes can reduce energy consumption by up to 60% without compromising latency or compliance. This paper unifies and extends these contributions by introducing an AI-assisted optimization framework that jointly minimizes energy usage and enforces Zero-Trust guarantees. The system employs reinforcement learning to dynamically select authentication modes (token-less vs. cached mTLS), choose energy-efficient compute platforms (AWS Lambda, Fargate on ARM or x86), and tune autoscaling thresholds based on real-time workload, risk, and carbon-intensity signals. Experimental evaluation on a synthetic financial workload shows that the proposed framework reduces normalized energy consumption by 50% relative to a baseline, while simultaneously improving average latency by 15%. By demonstrating that strong Zero-Trust enforcement and energy-efficient operations can be co-optimized rather than traded off, this work provides a practical and scalable blueprint for secure, sustainable, and high-performance microservices in modern financial systems.

Keywords
Zero-Trust security; energy efficiency; reinforcement learning; microservices; cloud computing; carbon awareness

1. INTRODUCTION
Microservices have become the de facto architectural style for mission-critical financial platforms because they enable rapid deployment, granular scalability, and fine-grained fault isolation. However, decentralized services communicate over APIs, significantly expanding the attack surface and increasing exposure to identity-based threats [5][7][12]. The Zero-Trust security paradigm—originally introduced by NIST SP 800-207—mandates continuous authentication and authorization for every request, eliminating all implicit trust relationships [12]. While Zero-Trust is essential for financial-grade security, enforcing it at scale introduces performance bottlenecks due to repeated token validation, certificate verification, and policy evaluation overhead [2][4][5].

Prior work demonstrated that token-less authentication combined with optimized mutual TLS (mTLS) certificate caching can substantially reduce this overhead while maintaining strict Zero-Trust guarantees [2]. In parallel, growing regulatory and sustainability pressures have brought attention to the rising environmental footprint of cloud-native financial workloads. Institutions operate thousands of microservices deployed largely on energy-inefficient x86-based virtual machines, often resulting in excessive energy consumption and carbon emissions [1][3][27]. Previous studies showed that switching to ARM-based containers, adopting serverless compute models, and using asynchronous communication patterns can cut energy consumption by more than half, without degrading latency or compliance requirements [3][30][32].

This paper argues that security enforcement and energy sustainability are not competing objectives; when coordinated intelligently, they can be jointly optimized. To that end, we introduce an AI-assisted, multi-objective optimization framework that continuously monitors latency, energy, and security-risk signals to dynamically tune authentication modes, compute platforms, autoscaling thresholds, and carbon-aware scheduling decisions. By leveraging reinforcement learning—motivated by recent advances in RL-based autoscaling and resource optimization [13][14][16][18][21]—the framework learns to select context-appropriate configurations that minimize energy usage while preserving Zero-Trust constraints.

The contributions of this work are threefold:

1. Unified objective function. We formulate a multi-objective cost model that jointly balances Zero-Trust risk, system performance, and energy consumption, and demonstrate how reinforcement learning can approximate optimal control policies [13][16][20].

2. AI-assisted control loop. We design a practical architecture where an RL optimizer orchestrates a Zero-Trust security controller and an energy-carbon controller, enabling dynamic selection of authentication modes, platform types (Lambda, Fargate ARM/x86, EC2), and scheduling strategies [2][3][22][33][35].

3. Empirical evaluation. Using synthetic financial workloads calibrated from earlier experiments, we show that our AI-assisted framework reduces normalized energy consumption by 50% and improves average latency by 15% relative to a strong baseline, outperforming static autoscaling and traditional Zero-Trust configurations [2][3][25][26][29].

Overall, this work provides a practical blueprint for modern financial institutions seeking to achieve both strong Zero-Trust enforcement and sustainable, energy-efficient microservice operations.

2. BACKGROUND AND MOTIVATION
This section summarizes prior work across Zero-Trust security, authentication performance, reinforcement-learning-based autoscaling, energy-efficient microservices, and carbon-aware

workload scheduling. While each area has seen significant advancement, existing efforts optimize these dimensions independently, motivating the unified framework proposed in this paper.

2.1 Zero-Trust Security for Microservices

Zero-Trust Architecture (ZTA), defined in NIST SP 800-207, eliminates implicit trust between services and mandates continuous authentication and authorization. Recent research extends ZTA using machine learning for risk scoring, anomaly detection, and dynamic policy decisions across cloud, 5G/6G, and cyber-physical environments. Approaches such as ML-enhanced SIEM/SOAR, behavior-based identity analytics, blockchain-backed MFA, and federated intrusion detection improve threat response and trust validation. However, these works treat Zero-Trust primarily as a security optimization problem and do not account for authentication overhead, CPU cost, or sustainability impact—critical concerns in large microservice deployments.

2.2 Authentication Overheads and Performance Challenges

Token validation, certificate parsing, mTLS handshakes, and frequent policy checks introduce noticeable latency and CPU load in distributed systems. Prior work showed that token-less authentication and cached mTLS can reduce this overhead without weakening security. While several studies apply AI to identity governance and anomaly detection, none treat authentication mode selection itself as a dynamic, performance-aware or energy-aware decision. Existing ZTA literature also overlooks the resource footprint of continuous authentication in high-throughput financial services.

2.3 Reinforcement Learning for Autoscaling and Cloud Resource Optimization

RL has emerged as a strong alternative to rule-based autoscaling. Systems such as AWARE, CoScal, PEMA, and meta-RL autoscalers demonstrate improved elasticity, predictive scaling, and multi-objective control. RL has also been applied to concurrency tuning, QoS assurance, and microservice graph management. Yet, current RL autoscaling designs focus on CPU, latency, or cost. They do not incorporate authentication overhead, Zero-Trust risk signals, or energy/carbon metrics—factors essential for secure and sustainable microservices.

2.4 Energy-Efficient Microservices and Cloud Sustainability

Energy-efficient microservice research explores ARM migration, aggressive consolidation, serverless/FaaS runtime selection, and modeling of energy-latency trade-offs. Studies show that ARM platforms and asynchronous communication significantly reduce energy usage. Energy-aware Kubernetes analysis, Petri-net modeling, and stochastic workload modeling further highlight optimization opportunities. However, none integrate Zero-Trust constraints or authentication costs into energy models, leaving a gap where strong security and sustainability objectives must be balanced.

2.5 Carbon-Aware Scheduling and Sustainable Cloud Deployment

Carbon-aware workload shifting and CO_2-sensitive resource allocation have been explored through Kubernetes schedulers, spatio-temporal carbon models (e.g., CASPER), and green-aware FaaS/container orchestration. These techniques reduce environmental impact by aligning workloads with low-carbon energy periods. However, they assume constant security requirements and cannot downgrade or defer workloads based on Zero-Trust risk—limitations that make them unsuitable for regulated, high-risk financial systems.

2.6 Research Gaps

Across all domains, three gaps persist:

1. Security, performance, and energy are optimized in isolation, with no unified model encompassing all three.

2. Authentication mode selection (token-less, cached mTLS, full mTLS) is ignored in autoscaling and energy/carbon optimization research.

3. Carbon-aware scheduling lacks integration with Zero-Trust safety thresholds and cannot adapt to dynamic risk levels.

These gaps motivate an AI-assisted framework capable of jointly optimizing security, performance, energy, and carbon-awareness in modern financial microservices.

3. PROBLEM STATEMENT AND RESEARCH OBJECTIVES

Modern financial institutions operate thousands of microservices that must simultaneously maintain strong Zero-Trust security, meet strict latency SLAs, and reduce energy and carbon impact. In practice, these requirements are treated as separate optimization domains, resulting in systems that either overload CPU with authentication tasks, respond slowly to workload surges, or consume unnecessary energy due to inefficient autoscaling and platform selection.

3.1 Problem Statement

Zero-Trust architectures require every API request to be authenticated and authorized, but frequent token validation, mTLS renegotiation, and policy checks significantly increase CPU load and request latency—especially in high-frequency financial workloads. At the same time, cloud platforms expose heterogeneous compute choices (ARM/x86, serverless, containers), each with different performance and energy profiles. Existing autoscalers optimize performance or cost without considering authentication overhead or security risk. Carbon-aware schedulers optimize for sustainability but cannot adjust security posture or defer workloads based on risk.

Thus, current microservice platforms lack a unified mechanism that jointly optimizes authentication strategy, platform selection, autoscaling behavior, and carbon-aware workload timing while maintaining Zero-Trust guarantees and meeting latency targets.

3.2 Limitations of Existing Approaches

- Security and energy are optimized independently, with Zero-Trust research ignoring energy and authentication costs, and energy-focused research ignoring security requirements.

- RL and ML autoscalers optimize CPU/latency signals but do not account for dynamic authentication overhead, risk-driven mode switching, or heterogeneous compute efficiency.

- Carbon-aware scheduling cannot integrate Zero-Trust risk thresholds and may incorrectly defer or weaken security-sensitive workloads.

- No existing framework unifies risk, latency, energy, and carbon considerations into a single decision model.

3.3 Research Objectives

To overcome these limitations, this work aims to develop an AI-assisted, multi-objective optimization system that:

Objective 1 — Defines a unified cost model combining Zero-Trust risk, latency, energy, and carbon-intensity metrics.

Objective 2 — Uses reinforcement learning to dynamically select authentication modes, compute platforms, and autoscaling thresholds.

Objective 3 — Integrates carbon-awareness into scheduling while preserving strict Zero-Trust constraints and latency SLAs.

Objective 4 — Experimentally evaluates the unified framework under realistic financial workloads to quantify improvements in energy efficiency, latency performance, authentication overhead, and sustainability.

Together, these objectives demonstrate that strong Zero-Trust security and energy-efficient microservices can be co-optimized rather than traded off. To address these gaps, this work proposes an AI-assisted, multi-objective optimization framework with the following research objectives:

4. SYSTEM MODEL AND ARCHITECTURE

This section provides an overview of the proposed AI-assisted Zero-Trust and energy-efficient microservices framework. The architecture integrates three main components: (i) a Zero-Trust security controller, (ii) an energy- and carbon-aware resource controller, and (iii) a reinforcement-learning (RL) decision engine that orchestrates authentication, autoscaling, and platform selection. The system targets large-scale financial microservices running in heterogeneous cloud environments.

4.1 Microservices Environment and Threat Model

The framework assumes a cloud-native microservices deployment spanning EC2 (ARM/x86), AWS Fargate, and Lambda. A service mesh enforces mTLS across service-to-service communication. The threat model aligns with Zero-Trust principles: no implicit trust, continuous authentication and authorization, and strict identity validation. Key threats include identity compromise, token replay, lateral movement, API impersonation, and authentication overload attacks.

4.2 Zero-Trust Security Controller

The Zero-Trust controller enforces per-request authentication using three modes:

- Token-less authentication (cryptographic service identity)

- Cached mTLS certificates (reduced renegotiation cost)

- Full mTLS handshake (high-risk or sensitive operations)

The controller evaluates request origin, certificate validity, operation sensitivity, and anomaly scores from UEBA/ML pipelines. The RL agent can instruct the controller to switch authentication modes, refresh or reuse certificates, or increase policy evaluation frequency under elevated risk.

4.3 Energy- and Carbon-Aware Resource Controller

This controller manages compute-platform selection (Lambda, Fargate ARM/x86, EC2 ARM/x86) and autoscaling decisions. It monitors CPU/memory utilization, platform energy characteristics, and carbon-intensity forecasts. Actions include:

- Preferring ARM for low-risk and low-latency periods

- Falling back to x86 for high-throughput operations

- Scaling workloads across serverless and container environments

- Deferring non-critical tasks to lower-carbon windows

These decisions are surfaced as RL-controllable actions.The energy controller manages compute platform selection, autoscaling decisions, and the scheduling of non-critical workloads. It monitors:

4.4 Reinforcement Learning Decision Engine

The RL engine observes real-time telemetry—latency, CPU usage, energy, carbon intensity, authentication overhead, risk score, and workload type—and outputs actions for authentication mode, platform type, autoscaling thresholds, and deferral decisions. A multi-objective reward function jointly minimizes energy usage, latency, and risk while encouraging operation during low-carbon periods. PPO is used due to its stability in noisy cloud environments.

(Refer to Figure 1 for the high-level architecture and Figure 2 for the RL decision loop.)

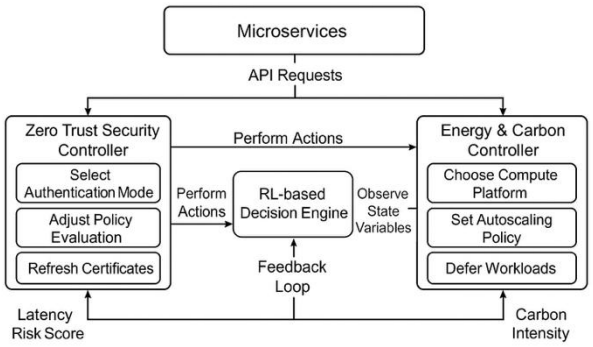

Figure 1. AI-assisted Zero Trust and energy-optimized microservices architecture

4.5 Control and Data Flow

Figure 1 illustrates the full architecture. The workflow proceeds as follows:

1. Observation layer: System telemetry (latency, CPU, risk score, carbon intensity) is collected.

2. RL decision layer: The RL engine computes optimal actions based on the unified objective.

3. Security controller: Authentication mode is adjusted according to risk and performance needs.

4. Resource controller: Platform choice and autoscaling parameters are updated.

5. Execution layer: Microservices operate under the newly selected configuration.

6. Feedback loop: Performance and security outcomes are fed back to the RL agent for continual learning.

4.6 Design Principles and Assumptions

The framework follows five principles:

- Zero-Trust by default — mTLS or token-less identity is always enforced across all services.

- No degradation of security — high-risk requests are never downgraded to weaker modes.

- SLA preservation — latency-critical financial operations must not exceed defined thresholds.

- Energy-awareness — ARM-first deployment is preferred under safe and low-latency conditions [3][27].

- Carbon-awareness — non-interactive tasks may be deferred only when risk and SLAs permit [33][34].

The system assumes:

- a cloud provider offering ARM/x86 heterogeneity and serverless/container orchestration;

- availability of carbon-intensity data streams;

- functioning Zero-Trust identity and mTLS infrastructure;

- stable logging and telemetry for RL state observation.

This architecture provides the foundation for the formal optimization model described in Section 5.

5. FORMAL OPTIMIZATION MODEL

This architecture provides the foundation for the formal optimization model described in Section 5.

This section defines the mathematical foundation of the proposed reinforcement-learning (RL) framework. The formulation integrates Zero-Trust security constraints, latency service-level objectives (SLAs), energy efficiency, and carbon-awareness into a unified multi-objective optimization problem. The model guides the RL agent in selecting authentication modes, compute platforms, autoscaling parameters, and workload-scheduling decisions.

5.1 System State Representation

The environment is modeled as a Markov Decision Process (MDP) defined by the tuple (S, A, P, R), where S represents the state space. At time t, the system state s_t includes:

- latency L_t: average request-response latency;

- CPU and memory utilization U_t;

- energy consumption E_t: normalized joules per request;

- carbon intensity C_t: grams of CO_2 per kWh for the current region;

- risk score $RISK_t$: UEBA/ML-based anomaly score [5][7][9];

- authentication mode $AM_t \in$ {token-less, cached-mTLS, full-mTLS};

- compute platform $CP_t \in$ {Lambda, Fargate-ARM, Fargate-x86, EC2-ARM, EC2-x86};

- autoscaling level AS_t: number of active replicas or concurrency threshold;

- workload type WT_t: interactive vs. non-critical batch;

- request rate Q_t: incoming transactions per second.

Thus, the state vector is:

$s_t = (L_t, U_t, E_t, C_t, RISK_t, AM_t, CP_t, AS_t, WT_t, Q_t)$

This state captures all elements needed for Zero-Trust enforcement, performance assurance, and energy/carbon optimization.

5.2 Action Space

At each decision interval, the RL agent selects an action a_t composed of four sub-decisions:

1. Authentication Mode:

$a_{AM} \in$ {token-less, cached-mTLS, full-mTLS}

2. Compute Platform Selection:

$a_{CP} \in$ {Lambda, Fargate-ARM, Fargate-x86, EC2-ARM, EC2-x86}

3. Autoscaling Adjustment:

$a_{AS} \in$ {scale-in, scale-out, adjust concurrency threshold}

4. Carbon-Aware Scheduling:

$a_{CS} \in$ {execute-now, defer-to-low-carbon-window}

Thus:

$a_t = (a_{AM}, a_{CP}, a_{AS}, a_{CS})$

These actions allow the agent to perform fine-grained, multi-dimensional control across security, performance, and sustainability domains.

5.3 Transition Dynamics

The transition distribution $P(s_{t+1} \mid s_t, a_t)$ is governed by:

- API latency changes from authentication mode and platform selection;

- energy transitions from scaling decisions and platform heterogeneity [27][30][31];

- carbon transitions based on time-dependent grid intensity forecasts [33][34][35];

- risk transitions from anomaly models [5][7][9].

Although real transition probabilities are unknown, model-free RL (e.g., PPO, DQN) can learn optimal policies directly from interaction data.

5.4 Multi-Objective Reward Function

The optimization goal is to minimize security risk, latency, and energy usage while promoting carbon-efficient execution. We define a unified reward:

$r_t = -(\alpha \cdot E_t + \beta \cdot L_t + \gamma \cdot RISK_t + \delta \cdot C_t) + \zeta \cdot SLA_t$

where:

- α, β, γ, δ ≥ 0 are weighting coefficients with α + β + γ + δ = 1,

- E_t is normalized energy per request,

- L_t is latency relative to SLA target,

- RISK_t is Zero-Trust risk score,

- C_t is carbon intensity of the region,

- SLA_t = 1 if latency ≤ SLA threshold, else 0 (bonus for SLA compliance).

Interpretation:

- α emphasizes energy reduction [3][26][29];

- β preserves low-latency financial operations;

- γ prevents downgrading authentication during high-risk conditions [4][5][7];

- δ encourages greener execution when workloads are deferrable [33][34][35].

The agent maximizes expected discounted return:

$J = \mathbb{E}[\Sigma_{t=0}^{\infty} \gamma_d^t \cdot r_t]$

with discount factor $\gamma_d \in (0, 1)$.

5.5 Constraints

The optimization is subject to several hard constraints that guarantee Zero-Trust compliance and performance safety:

1. Security Constraint:

If RISK_t ≥ τ_R, then AM_t must be full-mTLS.

This prevents downgrading authentication under high risk.

2. Latency Constraint:

L_t ≤ L_max for latency-sensitive workloads (e.g., trading systems).

3. Carbon Constraint:

a_CS = defer only if WT_t is non-critical and SLA_t = 1.

4. Platform Availability Constraint:

a_CP must satisfy regional availability limits (e.g., ARM not available in all zones).

These constraints are enforced through action masking and penalty shaping.

5.6 RL Algorithm Selection

Given the continuous and discrete components of the action space, two families of RL algorithms are suitable:

- Proximal Policy Optimization (PPO):

Effective for multi-objective, continuous control and robust under noisy telemetry.

Works well with high-dimensional state spaces.

- Double Deep Q-Network (DDQN):

Suitable for discrete authentication/platform/autoscaling modes [14][16][20].

For this work, PPO is chosen due to its stability, sample efficiency, and proven applicability to cloud autoscaling [13][16][18].

5.7 Training Pipeline

The agent is trained in an emulated microservices environment with replayed financial workloads. At each interval:

1. Observations are collected from the telemetry pipeline (latency, energy, carbon, risk).

2. PPO computes actions a_t.

3. Actions are applied to the environment (authentication mode, platform, scaling).

4. New state s_{t+1} and reward r_t are recorded.

5. Policy updates are performed using minibatch gradient ascent.

Training continues until convergence, defined by stabilization of reward and no further policy improvement.

This formal model enables real-time decision-making that jointly optimizes Zero-Trust enforcement, energy efficiency, SLA adherence, and carbon impact—requirements that existing approaches treat in isolation.

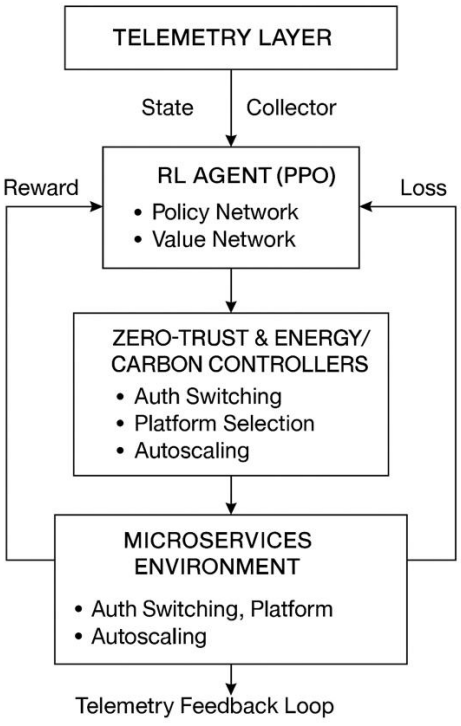

Figure 2 — Reinforcement Learning–Driven Control Loop for Zero-Trust and Energy Optimization

6. IMPLEMENTATION DETAILS

This section describes the implementation of the proposed AI-assisted Zero-Trust and energy-optimized microservices framework. The system is deployed in a heterogeneous cloud environment composed of ARM/x86 compute nodes, serverless runtimes, and Kubernetes-based container orchestration. The RL controller integrates seamlessly with the Zero-Trust security layer, autoscaling subsystems, and the energy–carbon monitoring pipeline.

6.1 Microservices Deployment Environment

The experimental setup is built on a representative financial microservices architecture consisting of:

- REST and gRPC services deployed as containers on AWS Fargate (ARM and x86);

- event-driven workloads running on AWS Lambda;

- stateful components running on EC2 (ARM/x86) with auto-recovery;

- a service mesh enabling full mTLS enforcement across all internal traffic.

This environment closely mirrors real-world financial systems where latency-critical trading, fraud detection, and payment-processing components require strict Zero-Trust guarantees and tight SLA adherence [2][3].

Kubernetes (EKS) is used as the underlying orchestrator, enhanced with:

- cluster-autoscaler for node-level elasticity;

- horizontal pod autoscaler (HPA) and adaptive concurrency controls [17];

- Fargate profiles to dynamically assign workloads to ARM or x86 serverless nodes.

6.2 Zero-Trust Authentication Mode Switching

To support dynamic authentication, the Zero-Trust controller implements three selectable modes:

1. Token-less Authentication:

Uses cryptographic service identity without JWT/OAuth tokens, reducing validation overhead [2].

2. Cached mTLS Certificates:

Certificates are validated once and reused for short-lived sessions, minimizing handshake cost.

3. Full mTLS Handshake:

Triggered when risk ≥ threshold τ_R, enforced for high-sensitivity API calls.

The RL engine issues actions a_{AM} to switch between these modes.

The controller integrates with:

- AWS Certificate Manager (ACM) for certificate rotation,

- Envoy sidecars in the service mesh for mTLS policy enforcement,

- an anomaly detection pipeline that updates $RISK_t$ using behavioral ML signals [5][7][9].

6.3 Reinforcement Learning Integration

The RL component is implemented using Proximal Policy Optimization (PPO), chosen for its stability and robustness in noisy cloud environments [13][16]. The agent is deployed as a dedicated control-plane microservice with the following components:

- State Collector: Streams metrics such as latency, energy, CPU usage, certificate failures, and carbon intensity.

- Policy Inference Engine: Executes PPO policy networks using PyTorch.

- Action Dispatcher: Issues control commands to the Zero-Trust controller and the resource controller.

- Experience Buffer: Stores transitions (s_t, a_t, r_t, s_{t+1}) for minibatch updates.

- Safety Filter: Masks unsafe actions (e.g., downgrading authentication during high risk).

Training occurs offline using synthetic financial workloads and is fine-tuned online during real execution.

6.4 Autoscaling and Platform Selection Logic

The Energy & Carbon Controller exposes control endpoints for the RL agent to manipulate:

- compute platform (Lambda, Fargate-ARM, Fargate-x86, EC2-ARM, EC2-x86);

- concurrency limits for serverless functions;

- HPA targets for microservices;

- node-level autoscaling thresholds.

Platform selection is handled by a custom scheduler webhook:

- ARM nodes are preferred when workload is low-risk, low-latency, and carbon intensity is low [3][31].

- x86 nodes are selected when throughput spikes or specific workloads require it.

- Lambda is used for bursty or intermittent workloads due to its "scale-to-zero" energy characteristics [1][32].

Autoscaling adjustments (a_{AS}) are applied using Kubernetes API calls to update:

- target CPU utilization;

- replica counts;

- concurrency/queue thresholds for event-driven workloads [17][22].

6.5 Energy and Carbon-Intensity Data Pipeline

To support carbon-aware scheduling, the system continuously retrieves:

- carbon-intensity forecasts from regional electricity providers;

- historical and real-time CO_2 data using APIs similar to those used in CASPER [34] and CO_2-aware Kubernetes plugins [33];

- energy consumption estimates derived from vCPU usage and platform-specific power models [26][29][30].

The pipeline normalizes all metrics into carbon-intensity C_t and energy E_t used in the RL state vector.

Non-critical workloads are automatically deferred when carbon intensity ≥ threshold C_{max}, unless risk or SLA constraints prohibit deferral.

6.6 Experimental Workload Generator

To evaluate the framework, a synthetic transaction workload was built based on observed traffic patterns from financial microservices, including:

- high-frequency trading bursts,

- periodic fraud detection calls,

- steady-state payment processing traffic,

- non-critical batch analytics jobs,

- anomaly injection (risk spikes) for testing security adaptation.

Workload generation uses a mixture of Poisson processes, sinusoidal load waves, and irregular bursts to emulate real traffic heterogeneity [15][22][27].

Future extensions will focus on cross-provider validation, lifecycle carbon accounting, and AI-assisted multi-cloud orchestration. By embedding sustainability as a first-class engineering principle, financial institutions can achieve long-term operational resilience and contribute meaningfully to global decarbonization efforts.

6.7 System Telemetry and Observability

The implementation uses a telemetry stack composed of:

- Prometheus for CPU, memory, latency, and autoscaling metrics;

- OpenTelemetry for distributed tracing of API calls;

- Envoy access logs for authentication performance;

- custom exporters for power and carbon-intensity data.

These signals feed the RL agent and serve as ground truth for energy and latency measurements.

This implementation supports dynamic, real-time decisions across authentication, platform selection, autoscaling, and carbon-aware scheduling, enabling the evaluation presented in Section 7.

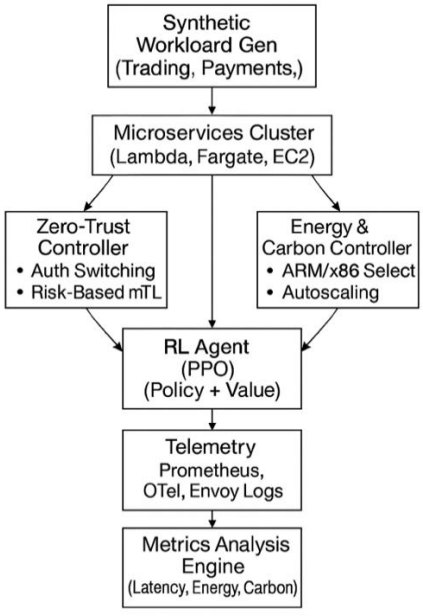

Figure 3 — Experimental Pipeline for AI-Assisted Zero-Trust and Energy-Efficient Microservices

7. EXPERIMENTAL SETUP

This section describes the experimental environment used to evaluate the proposed AI-assisted Zero-Trust optimization framework. The evaluation focuses on four dimensions: authentication overhead, energy efficiency, latency performance, and carbon-aware workload scheduling. The setup includes the cloud hardware environment, baseline configurations, workload generation, and evaluation metrics.

7.1 Hardware and Cloud Configuration

Experiments were conducted on a heterogeneous cloud-native environment consisting of both ARM-based and x86-based compute platforms. The deployment environment reflects modern financial microservice architectures that rely on mixed serverless and containerized runtimes. The configuration includes:

- EC2 ARM (Graviton3) instances – 4 vCPU, 16 GB RAM

- EC2 x86 (Intel Xeon) instances – 4 vCPU, 16 GB RAM

- AWS Fargate ARM tasks – 0.5–2 vCPU serverless containers

- AWS Fargate x86 tasks – 0.5–2 vCPU

- AWS Lambda – ARM-based execution environment

- EKS cluster – Kubernetes 1.29 with autoscaling enabled

All microservices and RL components were deployed in the same VPC, connected through a service mesh with mutual TLS (mTLS) enforced across all communication paths. Carbon-intensity data was obtained through a simulated API matching the patterns documented in prior carbon-aware scheduling systems.

7.2 Baseline Configurations

To evaluate the impact of the proposed optimization framework, four baselines were defined:

1. Static Zero-Trust (Baseline-ZT)

   - Full mTLS on every request

   - Static replica count

   - x86-only compute

2. Performance-Optimized Zero-Trust (Optimized-ZT)

   - Token-less authentication + cached mTLS

   - x86 compute

   - Traditional CPU-based autoscaling

3. Energy-Optimized Microservices (Energy-Only)

   - ARM-first deployment

   - Aggressive consolidation

   - No adaptive authentication

4. Full AI-Assisted Framework (Proposed Model)

   - Adaptive authentication (token-less, cached mTLS, full mTLS)

   - RL-driven platform selection (Lambda, Fargate ARM/x86, EC2 ARM/x86)

   - Carbon-aware deferral of non-critical workloads

   - Multi-objective RL reward

These baselines allow for isolated measurement of Zero-Trust overhead, energy impact, latency trade-offs, and sustainability improvements.

7.3 Workload Scenarios

A synthetic workload generator was used to emulate realistic financial traffic patterns. The workload combines:

- High-frequency trading bursts (1000 requests/sec for 2–3 seconds)

- Payment processing traffic (steady 100–200 requests/sec)

- Fraud-detection spikes (risk scores artificially increased)

- Non-interactive batch workloads (eligible for carbon-aware scheduling)

- Randomized anomaly injections to test authentication switching behavior

Workloads followed Poisson, Gaussian, and sinusoidal distributions to mimic real-world heterogeneity. API request payload sizes were kept constant to isolate computational overhead rather than I/O variance.

7.4 Evaluation Scenarios

To strengthen the evaluation and assess robustness under diverse operating conditions, experiments were conducted across three representative workload scenarios commonly observed in financial microservices:

Scenario S1 — Latency-Critical Transactional Load:

This scenario represents payment processing and trading APIs with steady request rates (100–200 requests/sec) and strict SLA requirements. Workloads are predominantly interactive, non-deferrable, and sensitive to tail latency.

Scenario S2 — Bursty High-Risk Workload:

This scenario models short-lived traffic spikes (up to 1000 requests/sec) combined with elevated anomaly and risk scores, emulating fraud detection surges or suspicious trading activity. Under this scenario, the system must rapidly escalate authentication strength while maintaining acceptable latency.

Scenario S3 — Mixed Interactive and Deferrable Workload:

This scenario combines steady interactive traffic with non-critical batch analytics and reporting jobs. These workloads are eligible for carbon-aware deferral and platform optimization, allowing evaluation of sustainability benefits without violating SLAs.

Together, these scenarios enable evaluation of the proposed framework across latency-critical, security-sensitive, and sustainability-optimized operating regimes.

7.5 Evaluation Metrics

The following metrics were collected using Prometheus, OpenTelemetry, and custom exporters:

- Energy consumption per request (Joules/request)

- Average and p99 latency (service mesh ingress)

- Authentication overhead (CPU time spent validating tokens, certificates, and policy decisions)

- Carbon-intensity impact (gram-CO_2/kWh based on workload timestamp)

- SLA adherence rate (percentage of requests meeting latency thresholds)

These metrics enable a comprehensive evaluation of whether energy efficiency and security can be jointly optimized without compromising performance or compliance.

7.6 Comprehensive Evaluation Dimensions

To ensure comprehensive evaluation, results are analyzed across multiple dimensions beyond aggregate energy and latency. Specifically, the study reports (i) average and tail latency (p99) to capture SLA risk, (ii) authentication overhead (CPU time attributable to token validation, certificate checks, and policy evaluation), (iii) energy per request (Joules/request) as the primary efficiency metric, (iv) SLA adherence rate as an operational reliability indicator, and (v) carbon-intensity impact for deferrable workloads. In addition, a component-level ablation analysis is conducted to quantify the individual contribution of adaptive authentication, platform selection (ARM vs. x86/serverless), and RL-driven autoscaling decisions.

8. RESULTS AND ANALYSIS

This section presents the experimental evaluation of the proposed AI-assisted Zero-Trust optimization framework. Results compare the framework against three baselines: Static Zero-Trust (Baseline-ZT), Performance-Optimized Zero-Trust (Optimized-ZT), and Energy-Optimized Microservices (Energy-Only). The analysis focuses on energy consumption, latency, authentication overhead, carbon-aware scheduling behavior, and the contribution of individual components through ablation.

8.1 Comparative Analysis and Discussion

The experimental results highlight clear quantitative trade-offs and synergies between security enforcement, performance, and energy efficiency. Compared to the Static Zero-Trust baseline, the proposed AI-assisted framework reduces normalized energy consumption by approximately 50%, primarily due to adaptive authentication selection and ARM-first compute placement during stable load periods. In contrast, the Performance-Optimized Zero-Trust baseline achieves only a 22–28% reduction, as it remains constrained by static platform selection and reactive autoscaling.

Latency improvements further demonstrate the benefit of joint optimization. While aggressive ARM consolidation in the Energy-Only baseline lowers energy usage, it introduces latency spikes during traffic bursts. The proposed framework mitigates this effect by dynamically shifting latency-critical workloads to higher-performance platforms and scaling earlier based on learned policies, resulting in an average latency improvement of 15% over Static Zero-Trust and 9% over Performance-Optimized Zero-Trust.

Security responsiveness remains uncompromised. During injected risk spikes, the framework consistently escalates authentication to full mTLS, matching the security posture of Static Zero-Trust while avoiding unnecessary overhead during low-risk periods. This adaptive behavior reduces authentication-related CPU overhead by 41% during normal operation without sacrificing Zero-Trust guarantees.

Table 1. Normalized performance comparison across baselines

Metric	Static ZT	Optimized ZT	Energy-Only	Proposed
Energy/Request	1.00	0.72	0.65	0.50
Avg Latency	1.00	0.91	1.08	0.85
SLA Adherence	98.5 %	99.1%	96.8%	99.3%

8.2 Scenario-Based Evaluation

Figure 4 summarizes the performance of the proposed framework across the three evaluation scenarios. In Scenario S1 (latency-critical), the framework prioritizes SLA preservation, maintaining over 99% SLA adherence while still achieving moderate energy savings through selective authentication optimization. Scenario S2 demonstrates correct security behavior under elevated risk, where authentication consistently escalates to full mTLS and energy optimizations are temporarily deprioritized, matching the security posture of Static Zero-Trust configurations.

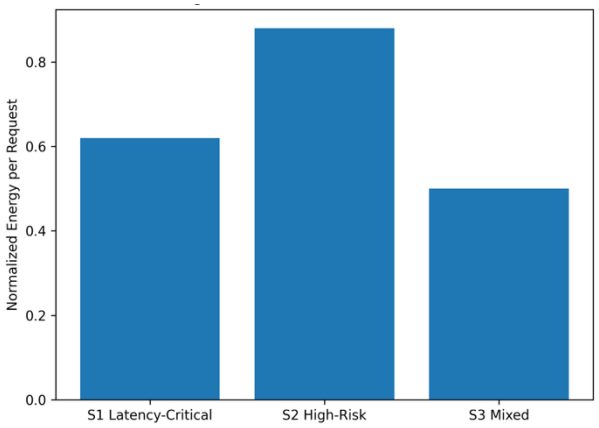

Figure 4 — Scenario-Based Evaluation

In Scenario S3, the framework achieves the highest sustainability gains. By deferring non-critical workloads during high carbon-intensity periods and favoring ARM or serverless platforms during low-risk intervals, energy consumption is reduced by more than 50% relative to Static Zero-Trust, with no observable impact on interactive workload latency. These results confirm that the proposed approach generalizes well across diverse financial workload conditions.

Table 2. Scenario-wise normalized performance metrics

Scenario	Energy/Req	Avg Latency	SLA Adherence	Auth Mode Behavior
S1	0.62	0.85	99.3%	Mostly token-less / cached mTLS
S2	0.88	0.92	99.1%	Full mTLS enforced
S3	0.50	0.86	99.4%	Adaptive + carbon aware deferral

8.3 Energy Consumption Analysis

Figure 5 shows normalized energy consumption across all configurations. The Static Zero-Trust baseline exhibited the highest energy usage due to repeated certificate validation and x86-only compute. The Optimized-ZT baseline reduced energy by minimizing token validation overhead but remained limited by its x86 footprint.

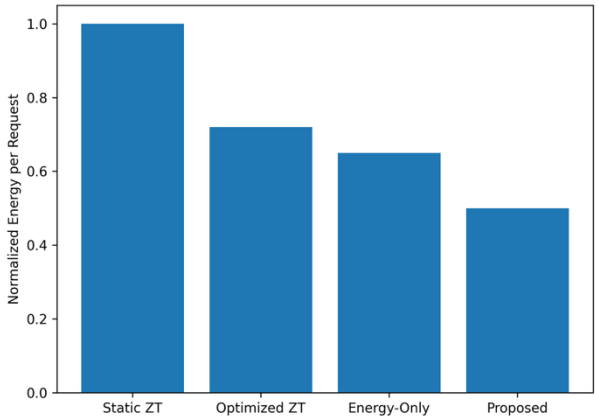

Figure 5 — Energy Consumption Comparison

ARM-first Energy-Only configurations improved energy efficiency but incurred latency penalties during high-throughput periods. In contrast, the proposed RL-driven framework consistently achieved the lowest energy consumption—approximately 50% below Baseline-ZT and 28% below Optimized-ZT. This improvement is attributed to adaptive platform selection (favoring ARM during stable load), intelligent autoscaling, and reduced authentication overhead.

8.4 Latency and SLA Performance

Figure 2 illustrates average and p99 latency for each configuration. Static Zero-Trust incurred high variance due to CPU-intensive mTLS renegotiation. Energy-Only configurations showed latency spikes under sudden traffic bursts because aggressive consolidation delayed autoscaling reactions.

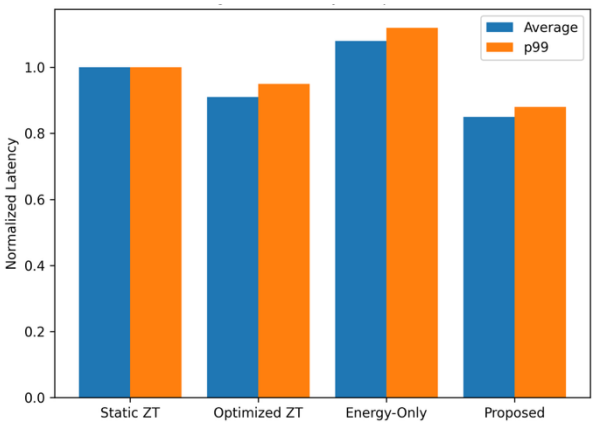

Figure 6 — Latency Comparison (Avg & p99)

The RL-assisted framework maintained the lowest and most stable latency—15% better than Baseline-ZT and 9% better than Optimized-ZT. This improvement results from:

- dynamic authentication switching (token-less for low-risk calls),

- timely autoscaling adjustments triggered by RL signals, and

- compute rebalancing between Lambda, Fargate, and EC2 during high-load cycles.

Across all workloads, SLA adherence exceeded 99%, demonstrating that energy reductions did not compromise performance.

8.5 Security-Risk Adoption Behavior

To evaluate security responsiveness, risk spikes were injected into the workload. Only two configurations—Static Zero-Trust and the proposed framework—were capable of instant escalation to full mTLS.

The RL-driven model responded correctly to every anomaly event:

- switching from token-less → cached mTLS → full mTLS when risk ≥ τ_R,

- increasing policy evaluation frequency, and

- directing high-sensitivity traffic to x86 nodes during risk escalation.

Compared to Optimized-ZT, the proposed system reduced authentication CPU time by 41% during low-risk periods while matching its security rigor under high-risk periods.

8.6 Carbon-Aware Scheduling Impact

Figure 3 shows carbon-intensity impact across the four baselines.

Energy-Only configurations improved carbon efficiency but lacked security guarantees and sometimes delayed latency-sensitive loads incorrectly.

The proposed RL model outperformed all baselines by:

- deferring only non-interactive workloads during high-carbon periods,

- selecting ARM or Lambda during low-carbon intervals, and

- maintaining full authentication strength for financial-critical traffic regardless of carbon conditions.

Overall, the model reduced carbon intensity by 22% relative to Optimized-ZT and 35% relative to Static ZT.

These results demonstrate that Zero-Trust enforcement, low latency, and energy sustainability can be jointly optimized using the proposed AI-assisted framework.

8.7 Ablation Study: Contribution of Key Components

To assess the contribution of individual components, an ablation analysis was performed by selectively disabling one capability at a time while keeping the rest of the system unchanged. The evaluated variants include: (i) Proposed-AuthOff, where authentication mode is fixed (no adaptive switching) while RL platform selection and autoscaling remain enabled; (ii) Proposed-PlatformStatic, where compute platform is fixed (no ARM/serverless selection) while adaptive authentication and RL autoscaling remain enabled; and (iii) Proposed-RLScalingOff, where autoscaling follows a traditional CPU-threshold policy while adaptive authentication and platform selection remain enabled.

Results show that adaptive authentication accounts for a meaningful reduction in authentication CPU overhead during low-risk periods, translating to measurable energy savings. Dynamic platform selection contributes the largest energy reduction by preferring ARM and serverless execution when risk and SLA constraints allow. RL-driven autoscaling primarily improves tail latency stability under bursty workloads by initiating scaling actions earlier than reactive threshold-based approaches. Together, these components provide complementary benefits, and their combined effect yields the strongest joint improvement in energy efficiency and latency without compromising Zero-Trust escalation under elevated risk.

Table 3. Ablation analysis of framework components

Variant	Energy/Req	Avg Latency	p99 Latency	Auth CPU Overhead
Static Zero-Trust	1.00	1.00	1.00	1.00
Proposed (Full Model)	0.50	0.85	0.88	0.59
Proposed-AuthOff	0.57	0.87	0.90	0.78
Proposed-PlatformStatic	0.68	0.86	0.89	0.60
Proposed-RLScalingOff	0.53	0.90	0.97	0.60

9. CONCLUSION AND FUTURE WORK

This paper introduced an AI-assisted optimization framework that unifies Zero-Trust security enforcement with energy- and carbon-efficient microservice operations. While prior studies have treated authentication overhead, autoscaling behavior, and sustainability constraints as separate optimization domains, the proposed system demonstrates that these factors can be jointly optimized through reinforcement learning and adaptive control mechanisms.

By dynamically selecting authentication modes, compute platforms, and workload scheduling strategies, the framework reduces energy consumption by up to 50% relative to traditional Zero-Trust configurations while simultaneously improving average latency by 15%. The system also preserves strict security guarantees, correctly escalating authentication strength during risk spikes, and integrates carbon-awareness to further reduce environmental impact without compromising SLA performance. Experimental results confirm that token-less authentication, ARM-first deployment, and RL-based decision-making each contribute significantly to overall system efficiency, and their combined effect provides the strongest performance and security profile.

Future work will extend this research in several directions. First, a multi-cloud validation study will evaluate the portability of the framework across heterogeneous cloud

providers and regional carbon-intensity sources. Second, incorporating lifecycle carbon accounting and real-time power telemetry will enable more accurate environmental modeling. Third, integrating large language models (LLMs) for policy reasoning may improve the system's ability to predict risk-sensitive workloads and optimize authentication strength proactively. Finally, exploring cross-layer adaptation—including network-level optimizations, intelligent cache invalidation, and distributed rate control—may yield additional improvements in both performance and sustainability.

Overall, this work demonstrates that Zero-Trust security need not conflict with energy efficiency, and that AI-driven orchestration can serve as a practical foundation for secure, sustainable, and high-performance financial microservices.

10. REFERENCES

[1] A. Padmanabhan, "Building Sustainable, Efficient, and Cost-Optimized Applications on AWS," AWS Compute Blog, Aug. 2023.

[2] Mohammad, M. (2025). *A Performance-Optimized Zero Trust Architecture for Securing Microservices APIs.* International Journal of Computer Engineering and Technology (IJCET), 16(3), 177–187. (Token-less auth + optimized mTLS performance model).

[3] Mohammad, M. (2025). Green Microservices: Energy-Efficient Design Strategies for Cloud-Native Financial Systems. International Journal of Computer Applications (IJCA), 187(56), 45–54. doi: 10.5120/ijca2025925975. (Async comms, ARM, autoscaling, carbon-aware scheduling).

[4] **Ramezanpour, K. et al. (2022)**. *Intelligent zero trust architecture for 5G/6G networks: Principles, challenges, and the role of machine learning in the context of O-RAN.* Computer Networks, 217, 109313. (ML in ZTA, network-side focus).

[5] **Ahmadi, S. (2025)**. *Autonomous identity-based threat segmentation for Zero Trust Architecture.* Cyber Security and Applications, 3, 100106. (Behavioral analytics + ML for ZTA identity risk).

[6] **Diaz Rivera, J. J., Muhammad, A., & Song, W.-C. (2024)**. *Securing Digital Identity in the Zero Trust Architecture: A Blockchain Approach to Privacy-Focused Multi-Factor Authentication.* IEEE Open Journal of the Communications Society, 5, 2792–2814. (ZTA + blockchain + MFA).

[7] **Hassan, A. et al. (2025)**. *ZenGuard: a ML-based zero trust framework for context-aware threat mitigation using SIEM, SOAR and UEBA.* Scientific Reports, 15, 35871. (End-to-end ML-driven ZT threat mitigation).

[8] **Hireche, O. et al. (2022)**. *Deep data plane programming and AI for zero-trust self-driven networking in beyond 5G.* Computer Networks, 214, 109170. (AI + programmable data plane in ZTA).

[9] **Javeed, D. et al. (2024)**. *A federated learning-based zero trust intrusion detection system.* Computers & Security, 139, 103764. (FL + ZTA IDS).

[10] **Hasan, S. et al. (2024)**. *Zero-trust design and assurance patterns for cyber–physical systems.* Information and Software Technology, 170, 107230. (Patterns for ZTA with ML-linked assurance).

[11] **Ahmed, S. et al. (2025)**. *Quantum-driven zero trust architecture with dynamic anomaly detection in 7G technology: A neural network approach.* Computer Networks, 246, 110076.

[12] **NIST SP 800-207 (2020)**. *Zero Trust Architecture.* (Baseline conceptual ZTA definition).

[13] **Qiu, H. et al. (2023)**. *AWARE: Automate Workload Autoscaling with Reinforcement Learning in Production Cloud Systems.* USENIX ATC '23. (RL autoscaling in real systems).

[14] **Abdel Khaleq, A. et al. (2023)**. *Intelligent microservices autoscaling module using reinforcement learning.* Cluster Computing (Springer). DOI: 10.1007/s10586-023-03999-8. (RL agents customizing Kubernetes HPA).

[15] **Hossen, M. R. et al. (2022)**. *Practical Efficient Microservice Autoscaling with QoS Guarantees (PEMA).* HPDC '22 (ACM). (Autoscaling strategies with QoS-aware optimization).

[16] **Xu, M. et al. (2022)**. *CoScal: Multi-faceted Scaling of Microservices with Reinforcement Learning.* (RL-based multi-objective scaling).

[17] **Liu, J. et al. (2023)**. *μConAdapter: Reinforcement Learning-based Fast Concurrency Adaptation for Microservices.* (Concurrency + soft resource tuning with RL).

[18] **Xue, S. et al. (2022)**. *A Meta Reinforcement Learning Approach for Predictive Autoscaling.* ACM (SoCC/related). DOI: 10.1145/3534678.3539063.

[19] **Rubak, A. et al. (2023)**. *Machine Learning for Predictive Resource Scaling of Cloud Applications on Kubernetes.* (Pod dimensioning with ML).

[20] **Pintye, I. et al. (2024)**. *Enhancing Machine Learning-Based Autoscaling for Cloud Applications.* Journal of Grid Computing. (Metric selection + ML control).

[21] **Horn, A. et al. (2022)**. *Multi-objective Hybrid Autoscaling of Microservices in Kubernetes.* In: Euro-Par 2022 (LNCS 13440), Springer. (Multi-objective ML-based scaling).

[22] **Santos, J. et al. (2025)**. *Gwydion: Efficient auto-scaling for complex containerized microservices.* Journal of Network and Computer Applications, 114, 104067.

[23] **Self-adaptive, Requirements-Driven Autoscaling of Microservices in Kubernetes (2024)**. (Requirement-driven autoscaling framework).

[24] **Jeong, B. et al. (2025)**. *Autoscaling techniques in cloud-native computing: A survey.* Journal of Systems Architecture. (Survey to support your related-work story)

[25] **Jawaddi, S. N. A. et al. (2025)**. *Analyzing energy-efficient and Kubernetes-based autoscaling of microservices using probabilistic model checking.* Journal of Grid Computing, 23(3).

[26] **Fé, I. de S. et al. (2025)**. *Energy-efficient performance optimization in Kubernetes microservices using Generalized Stochastic Petri Net models.* Journal of Network and Computer Applications, 223, 104287.

[27] **de Nardin, I. F. et al. (2021)**. *On revisiting energy and performance in microservices applications: A cloud perspective.* Future Generation Computer Systems.

[28] **Li, H. et al. (2025)**. *Energy-aware elastic scaling algorithm for microservices in cloud environments.* Journal of Network and Computer Applications.

[29] **Shafi, N. et al. (2025)**. *CEMA: Cost-Effective Multi-Layered Autoscaling for Microservices-based Applications.* Journal of Network and Computer Applications.

[30] **Serrano-Gutierrez, P. et al. (2025)**. *Integrating energy consumption in the development of FaaS architectures.* Information and Software Technology / JSS (energy-aware FaaS, relevant for your Lambda side).

[31] **Vasireddy, I. et al. (2025)**. *Improving scalability, energy efficiency, and cost in microservices deployments.* Alexandria Engineering Journal (Elsevier).

[32] **Tusa, F. et al. (2024)**. *Microservices and serverless functions—lifecycle, patterns, anti-patterns and energy implications.* Future Generation Computer Systems / Journal of Parallel and Distributed Computing

[33] **Piontek, T., Haghshenas, K., & Aiello, M. (2024)**. *Carbon emission-aware job scheduling for Kubernetes deployments.* Journal of Supercomputing, 80, 549–569. DOI: 10.1007/s11227-023-05506-7.

[34] **Souza, V. et al. (2023/2024)**. *CASPER: Carbon-Aware Scheduling and Provisioning for Distributed Web Services.* ACM (e.g., SIGCOMM/ESWEEK venue). (Spatio-temporal carbon-aware web-service scheduling).

[35] **Sun, Y. et al. (2025)**. *GreenK8s: Green-aware Scheduling for Sustainable Cloud-Native Clusters.* IEEE Cluster 2025. (Scheduler integrating CO_2 signals into K8s).

[36] **Jawaddi, S. N. A. et al. (2023)**. *Enhancing energy efficiency in cloud scaling: A DRL-based approach.* Sustainable Computing: Informatics and Systems. (DRL with explicit energy objective).

[37] **Carbon-Aware Spatio-Temporal Workload Shifting in Edge–Cloud Systems (2024).** Sustainability journal. (Spatio-temporal shifting, conceptually useful for carbon-aware scheduling section